November 12, 1965

Dr. D. H. Lehmer
Department of Mathematics
University of California
Berkeley, California

Dear Dr. Lehmer:

I was delighted to see your chapter in "Applied Combinatorial Mathematics". I can only wish we had seen your work sooner; it might have saved several kilo-instructions of redundant code in the course of rediscovering some of the same algorithms. If you know of any other literature on these kinds of tricks, besides Iverson and your bibliography, I would be grateful to have reference to it.

LEHMER

To show how naive I could be, I "rediscovered" Gray code a couple of years ago in an exercise where I needed to generate the 2^n n-bit binary numbers (rather an equivalent vector) in the most orderly, recursively efficient way possible. Trouble was, I already knew Gray code but in an A-D, not a combinatorial, context!

For a permutation generator, don't you think the attached procedure would beat the methods you listed? I haven't coded it and would welcome your remark on its utility before attempting it. (In some heuristic chemistry problems we need to be able to start an exhaustive search of permutations beginning at some arbitrary form in the list.)

Professor Polya is a treat to talk to, but otherwise I have not found many combinatory enthusiasts here at Stanford. I enjoyed your chapter so much I look forward to meeting you sometime. Meanwhile I might burden you with some writings to explain some of the problems we are working on. I have a hunch you might understand why they are so intriguing to us.

Sincerely yours,

Joshua Lederberg
Professor of Genetics

P.S. Do you happen to know whether a tape of the permutations of 9 or 10 integers already exists, i.e., so I could get a copy of it?

Stanford Medical Research Note
65-GEN-1101

Algorithm (untested)

Recursive Generator of m! Permutations of m distinct items

Begin

Initialize by mapping the items on to the integers 1, 2, ... i ... m

= Si ; Si = i.

For p = 1(1)m-1:

We will use the already computed list of permutations of (p) items,

Lp, to generate the list of (p + 1) as follows.

For S_1 = 1(1)p + 1

Stack the remaining integers into a list S of length p.
Use Lp as a signature or mapping vector to produce all
permutations of these integers, which will form $S_2 \ldots . S_{p+1}$.

End

Comment. This procedure should buy some efficiency of calculation at the expense of storage and the cost of using the mapping vector. The list Lp does not have to be stored in its entirety if storage limitations preclude this; several digits' worth could be stored and the rest recalculated as needed.

On the other hand, Lp requires serial, not random, access, allowing efficient use of secondary storage. Lp for p odd and even should probably be on alternate tapes, as the Lp tape may have to be back spaced (p + 1) times while the L_{p+1} tape is written.